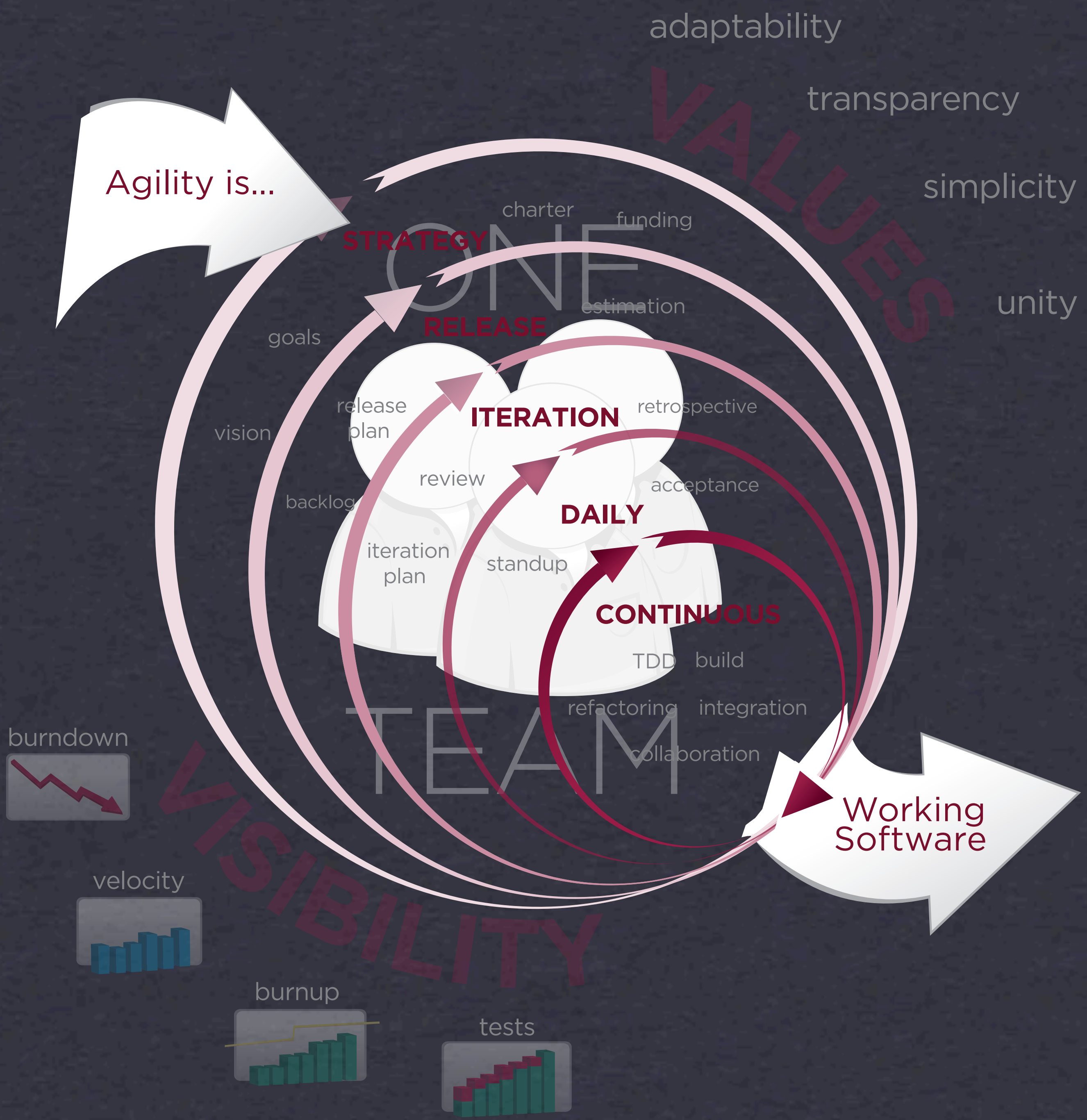
Material and some slide content from:
- Krzysztof Czarnecki
- Ian Sommerville



Development Processes

Reid Holmes

Lecture 18 - Tuesday November 16 2010.

NASA Case Study

- On-board shuttle group
  - 260 developers; 420 KLOC
  - 1 defect in each of the last three versions
- 4 flight control computers vote @ 250 Hz
  - 1 independent computer takes over in case of failure
- Process: Level 5 SEI CMM
  - Everything is about process.
  - ?
  - No change is made without corroborating documentation.

What Is a Software Engineering Process?

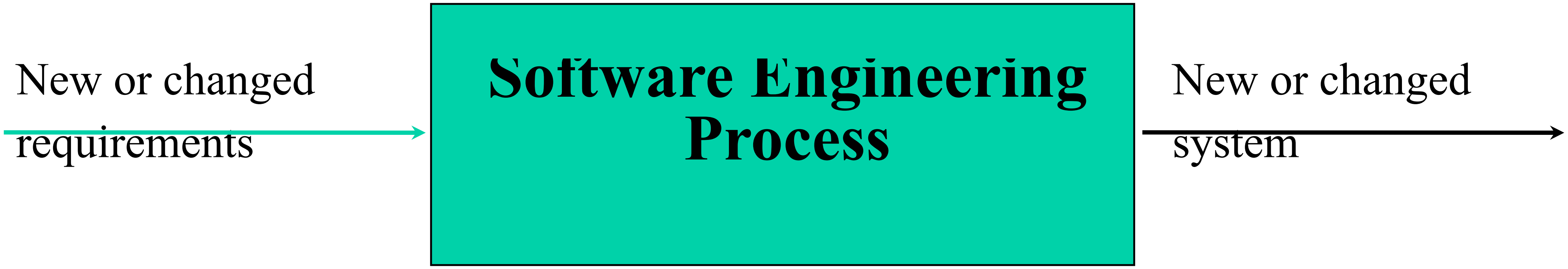
A process defines **Who** is doing **What**, **When** and **How** in the development of a software system

– Roles and workflows

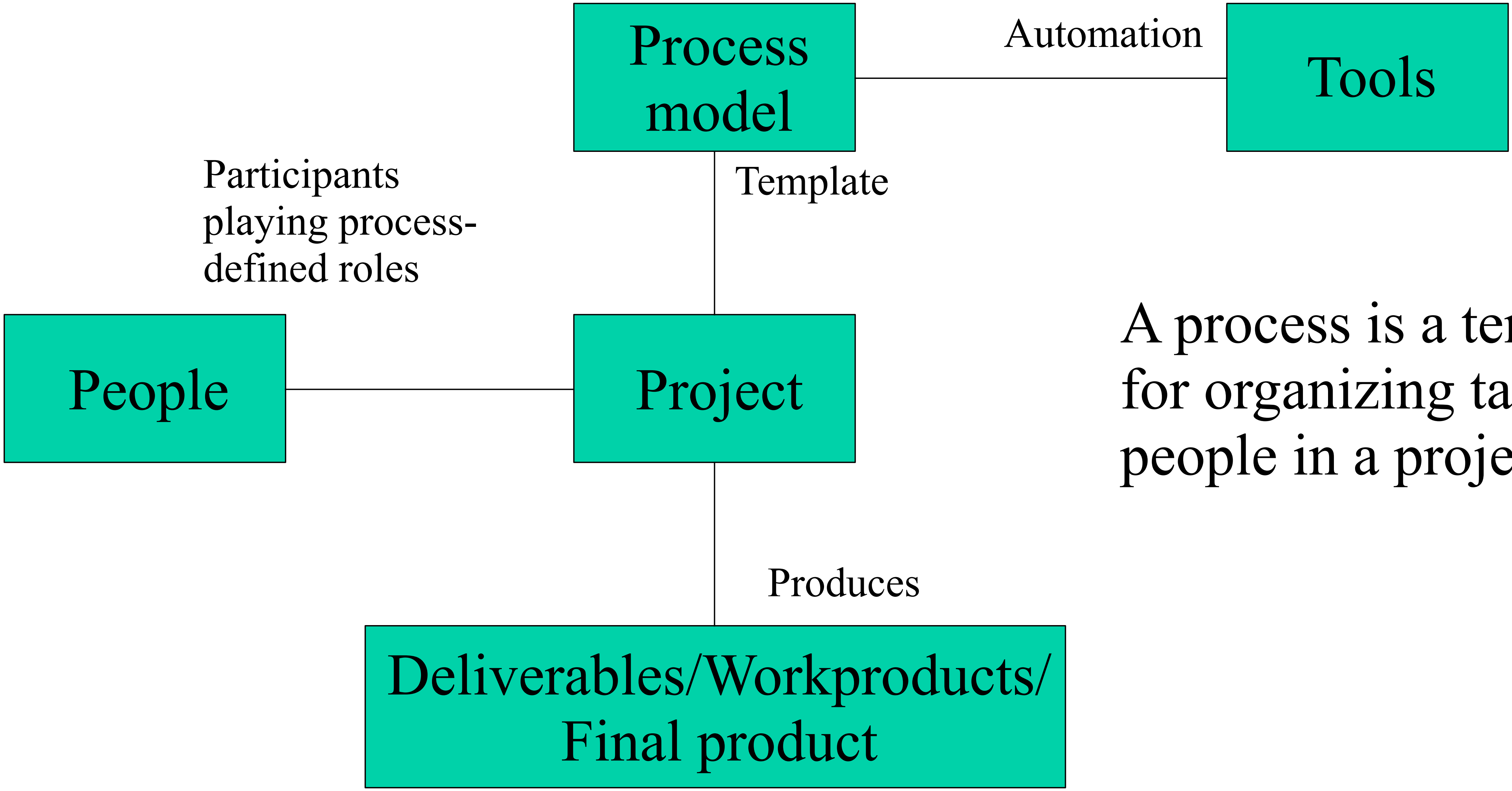
– Milestones

– Guideline

– …



Process vs. Product



Process model

Automation

Tools

Template

Participants playing process-defined roles

People

Project

Produces

Deliverables/Workproducts/ Final product

A process is a template for organizing tasks and people in a project

An Effective Process ...

- Provides guidelines for efficient development of quality software
- ?
- Captures and presents best practices
  - Learn from other’s experiences
  - Mentor on your desktop
  - Extension of training material
- Promotes common vision and culture
- Provides roadmap for applying tools
- Delivers information on-line, at your finger tips

Lightweight vs. Heavyweight Processes

Heavyweight
e.g., V-Process

Customizable Framework
e.g., Rational Unified Process (RUP)

Agile (Lightweight)
e.g., eXtreme Programming (XP)

Document driven
Elaborate workflow definitions
Many different roles
Many checkpoints
High management overhead
Highly bureaucratic

Focus on working code rather than documentation
Focus on direct communication (between developers and between developers and the customer)
Low management overhead

Process choice

- Process used should depend on type of product which is being developed
  - For large systems, management is usually the principal problem so you need a strictly managed process. For smaller systems, more informality is possible.
- High costs may be incurred if you force an inappropriate process on a development team

©Ian Sommerville 1995 [modified]

Waterfall

Spiral

Rational Unified Process (RUP)

- Iterative and incremental
- Use-case-driven
- Architecture-centric
- Uses UML as its modeling notation
- Process framework
  - Comprehensive set of document templates, process workflow templates, and process guidelines
  - Distributed by IBM
- Philippe Kruchten. The Rational Unified Process: An Introduction (3rd Edition). Addison-Wesley, 2003

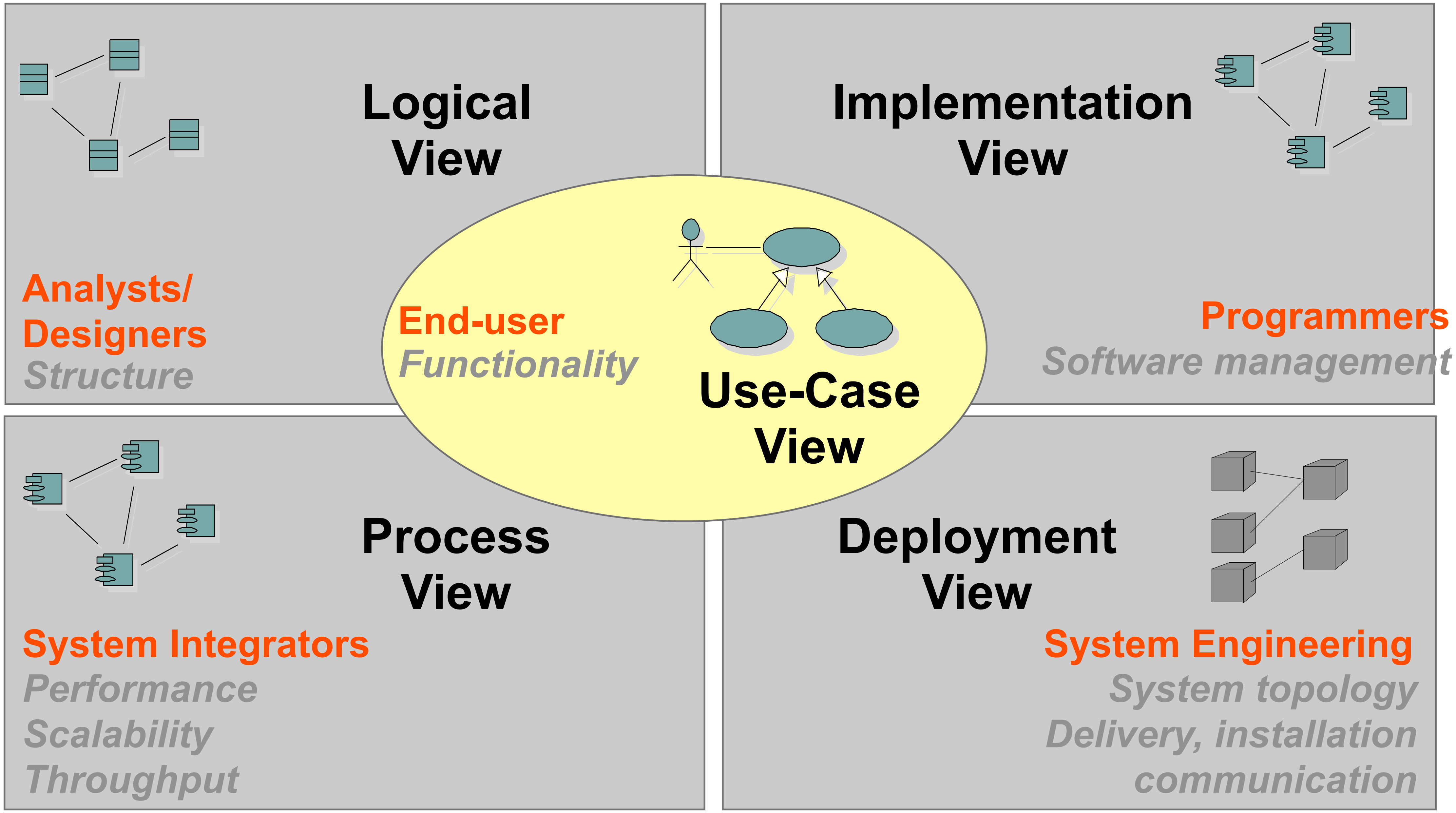
Rational Unified Process Is Use-Case-Driven

- Use cases are concise, simple, and understandable by a wide range of stakeholders
  - End users, developers and acquirers understand functional requirements of the system
- Use cases drive numerous activities in the process:
  - Creation and validation of the design model
  - Definition of test cases and procedures of the test model
  - Planning of iterations
  - Creation of user documentation
  - System deployment
- Use cases help synchronize the content of different models

Rational Unified Process Is Architecture-Centric

- Architecture is the focus of the elaboration phase
  - Building, validating, and baselining the architecture constitute the primary objective of elaboration
- The Architectural Prototype validates the architecture and serves as the baseline for the rest of development
- The Software Architecture Description is the primary artifact that documents the architecture chosen
- Other artifacts derive from architecture:
  - Design guidelines including use of patterns and idioms
  - Product structure
  - Team structure

Representing Architecture: The 4+1 View Model



Logical View

Analysts/ Designers

Structure

Implementation View

Programmers

Software management

End-user

Functionality

Use-Case View

Process View

System Integrators

Performance

Scalability

Throughput

Deployment View

System Engineering

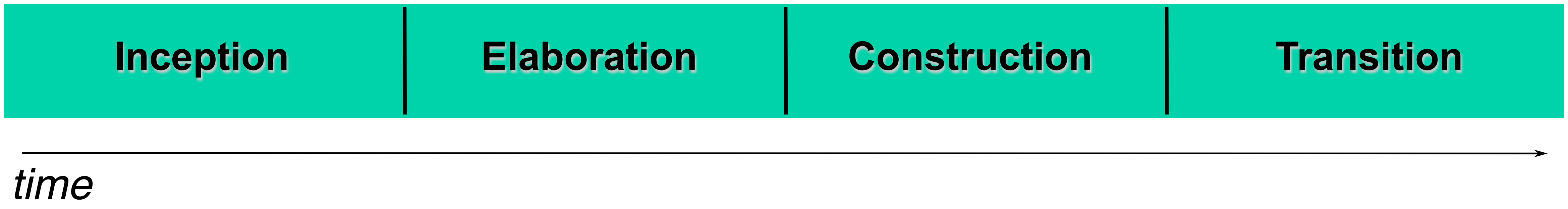
System topology

Delivery, installation

communication

Philippe Kruchten. The 4+1 View Model of Architecture. *IEEE Softw.* 12, 6 (Nov. 1995), 42-50

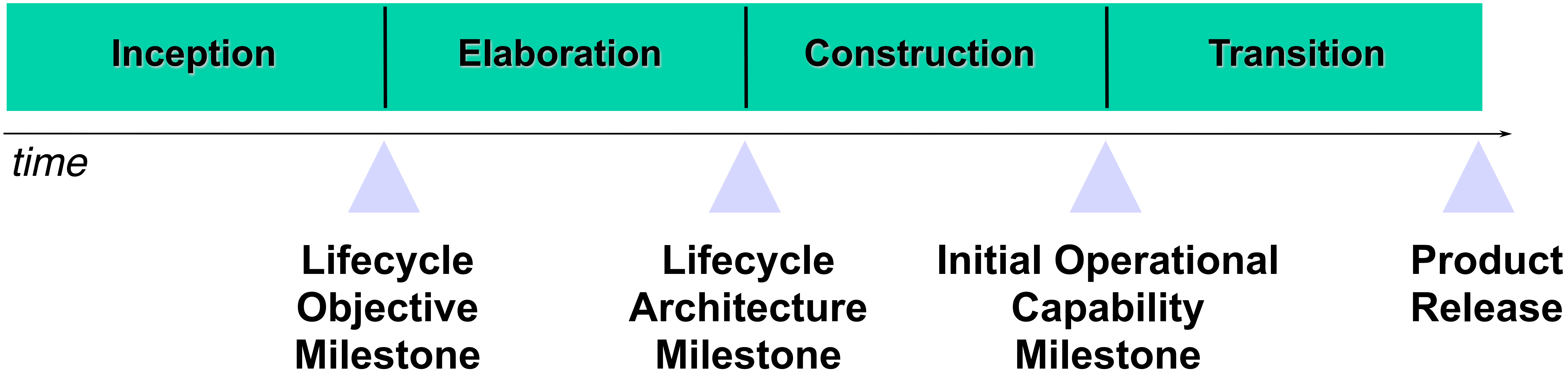
Process Architecture - Lifecycle Phases



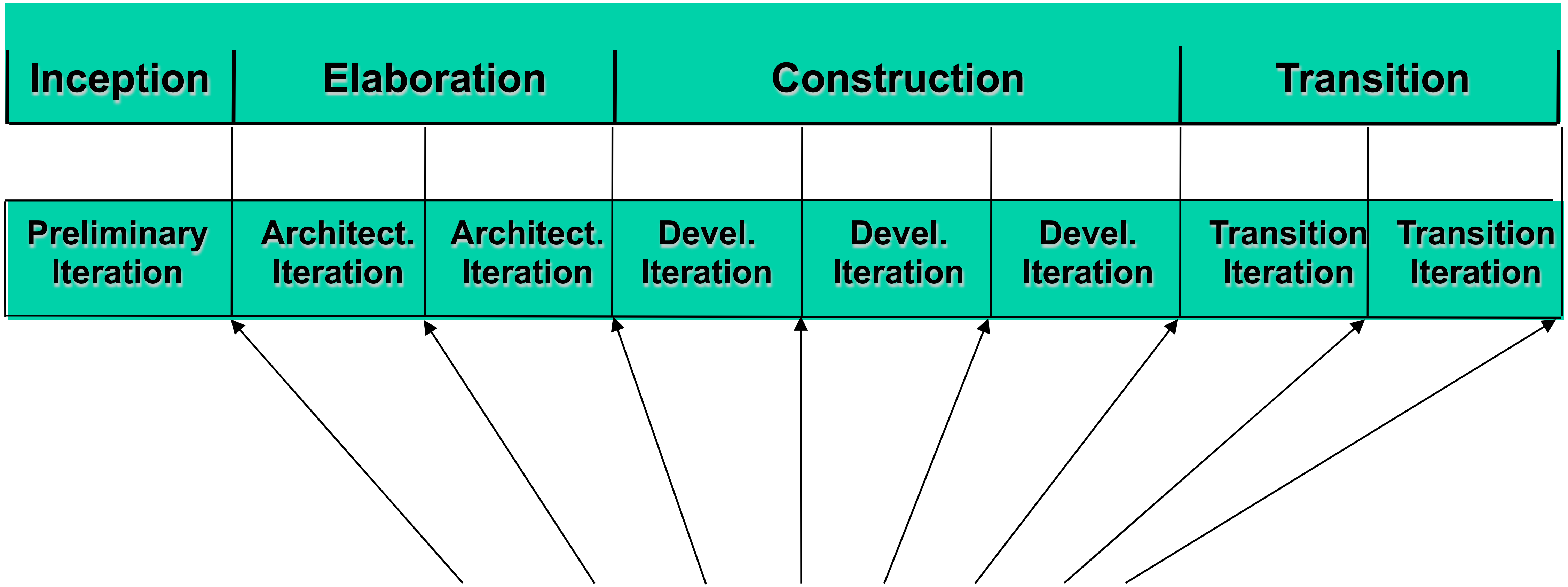
The Rational Unified Process has four phases:

- **Inception** - Define the scope of project
- **Elaboration** - Plan project, specify features, baseline architecture
- **Construction** - Build the product
- **Transition** - Transition the product into end user community

Phase Boundaries Mark Major Milestones


Inception
Elaboration
Construction
Transition
time
Lifecycle Objective Milestone
Lifecycle Architecture Milestone
Initial Operational Capability Milestone
Product Release

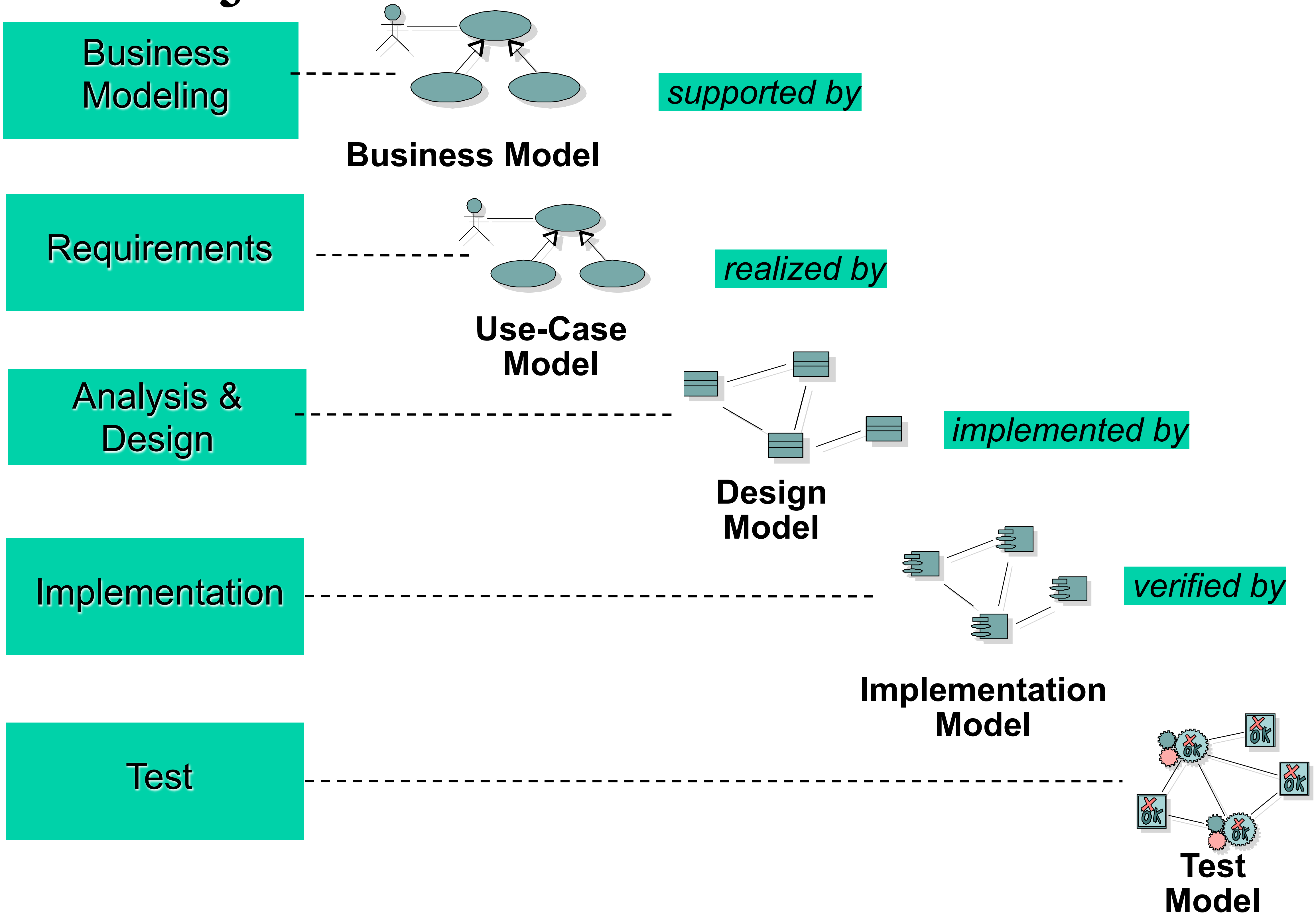
Iterations and Phases



Minor Milestones: Releases

An iteration is a distinct sequence of activities with an established plan and evaluation criteria, resulting in an executable release (internal or external)

Major Workflows Produce Models



Business Modeling

Business Model

supported by

Requirements

Use-Case Model

realized by

Analysis & Design

Design Model

implemented by

Implementation

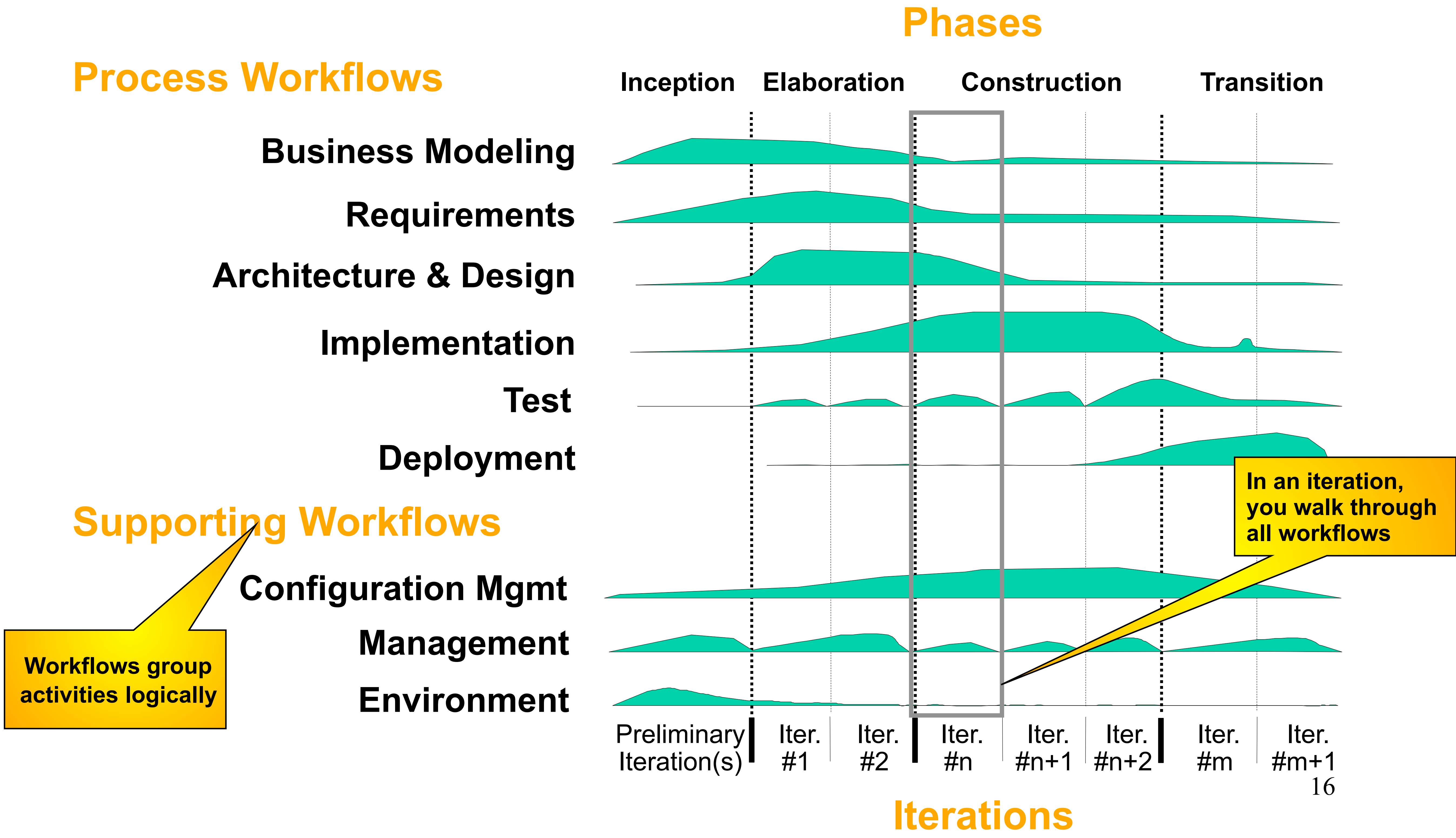
Implementation Model

verified by

Test

Test Model

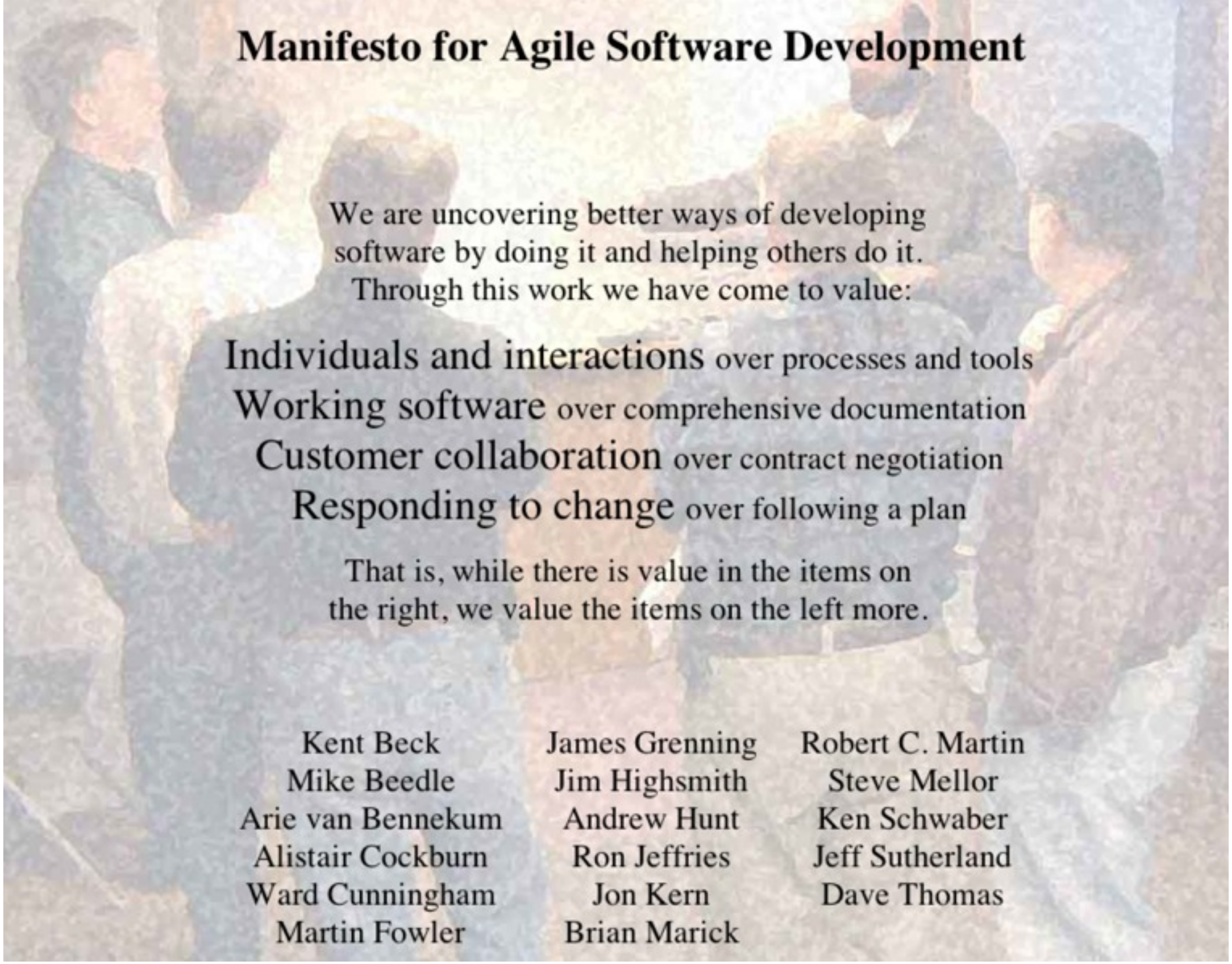
RUP Overview



In an iteration, you walk through all workflows

Workflows group activities logically

agilemanifesto.org



Manifesto for Agile Software Development

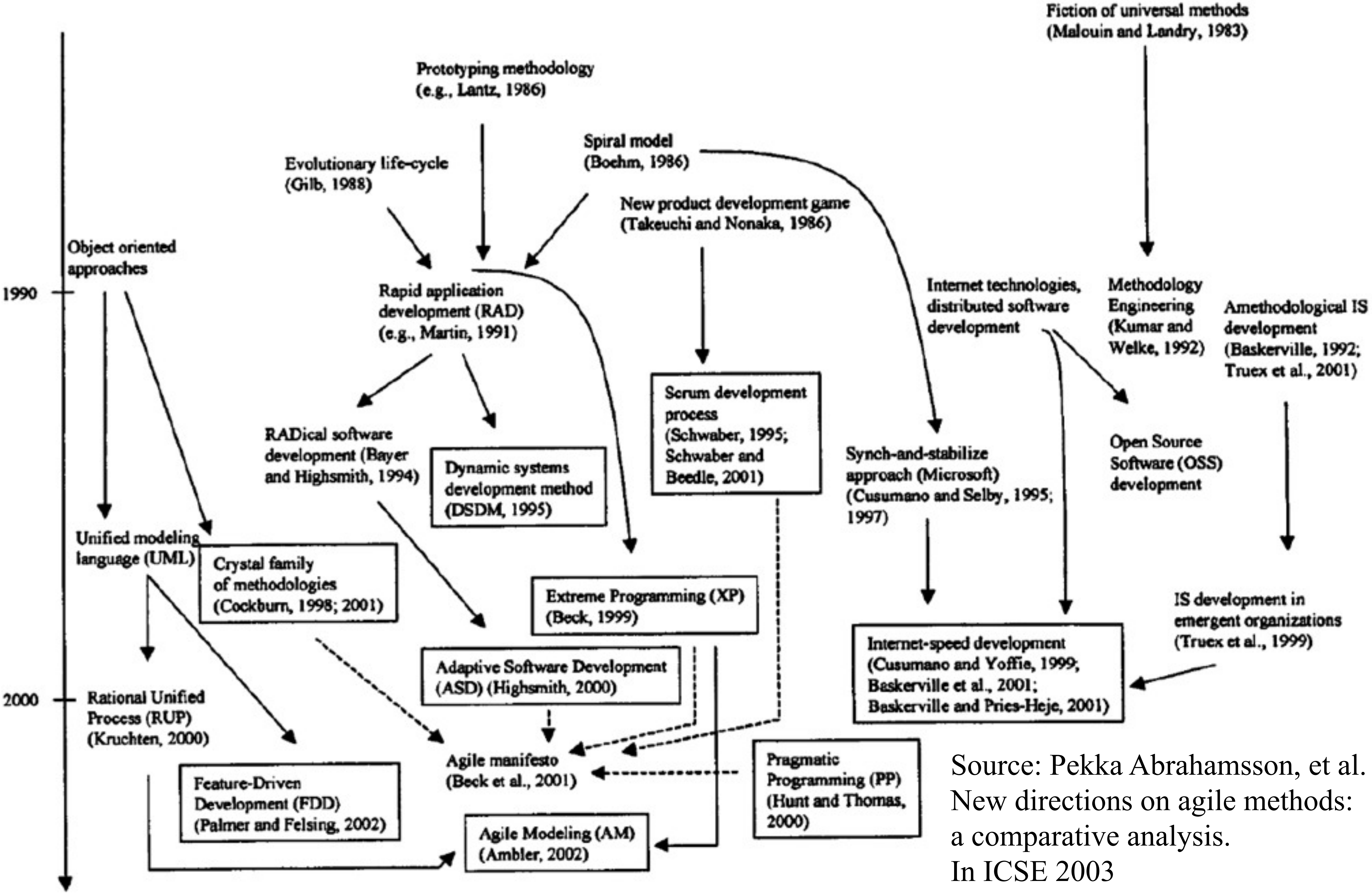
We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

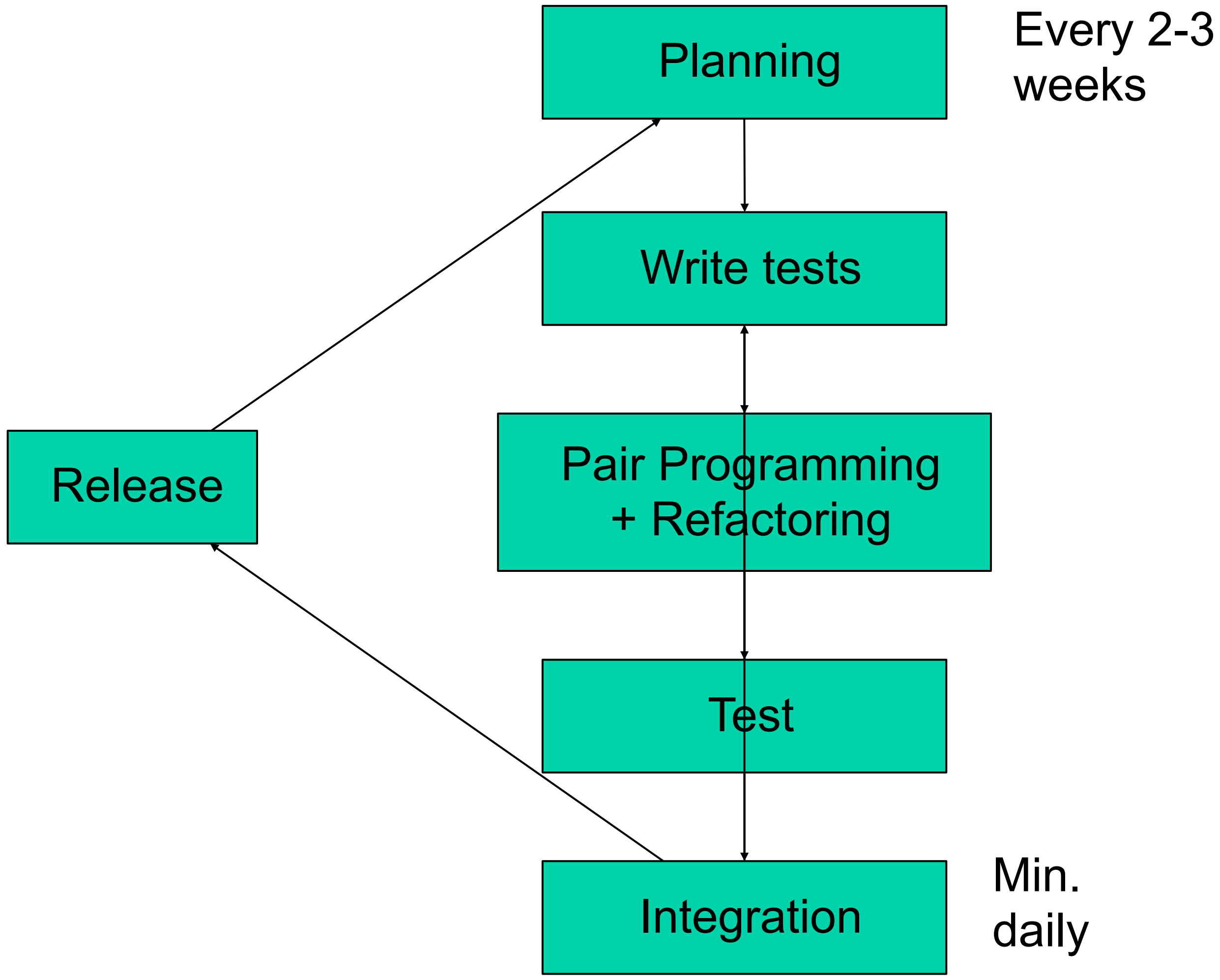
History of Agile Methods



1990
2000
Fiction of universal methods (Malouin and Landry, 1983)
Prototyping methodology (e.g., Lantz, 1986)
Spiral model (Boehm, 1986)
Evolutionary life-cycle (Gilb, 1988)
New product development game (Takeuchi and Nonaka, 1986)
Object oriented approaches
Rapid application development (RAD) (e.g., Martin, 1991)
Internet technologies, distributed software development
Methodology Engineering (Kumar and Welke, 1992)
Amethodological IS development (Baskerville, 1992; Truex et al., 2001)
Scrum development process (Schwaber, 1995; Schwaber and Beedle, 2001)
RADical software development (Bayer and Highsmith, 1994)
Dynamic systems development method (DSDM, 1995)
Synch-and-stabilize approach (Microsoft) (Cusumano and Selby, 1995; 1997)
Open Source Software (OSS) development
Unified modeling language (UML)
Crystal family of methodologies (Cockburn, 1998; 2001)
Extreme Programming (XP) (Beck, 1999)
IS development in emergent organizations (Truex et al., 1999)
Adaptive Software Development (ASD) (Highsmith, 2000)
Internet-speed development (Cusumano and Yoffie, 1999; Baskerville et al., 2001; Baskerville and Pries-Heje, 2001)
Rational Unified Process (RUP) (Kruchten, 2000)
Agile manifesto (Beck et al., 2001)
Pragmatic Programming (PP) (Hunt and Thomas, 2000)
Feature-Driven Development (FDD) (Palmer and Felsing, 2002)
Agile Modeling (AM) (Ambler, 2002)

Source: Pekka Abrahamsson, et al.
New directions on agile methods:
a comparative analysis.
In ICSE 2003

Extreme Programming (XP)



Characteristics

- Evolutionary development
- Collection of 12 „Best Practices“
- ?

- Testing is a crucial element of the process
- Focus on flexibility and efficiency of the process
- Designed for small teams (<10)

XP Practices (I)

- The planning game
  - Stakeholder meeting to plan the next iteration
  - Business people decide on business value of features
  - Developers on the technical risk of features and predicted effort per feature
- Small releases
  - Start with the smallest useful feature set; release early and often, adding a few features each time
- Metaphor
  - Each project has an organizing metaphor, a providing easy to remember naming conventions
  - http://www.agile2005.org/RP1.pdf

XP Practices (II)

- Simple Design
  - Always use the simplest possible design that gets the job done (runs the tests and states intentions of the programmer)
  - ?
- Testing
  - ?
  - Programmers write unit tests and customers write acceptance tests
- Refactoring
  - Refactoring is done continuously; the code is always kept clean

XP Practices (III)

- Pair programming
  - All production code written by two programmers
  - One programmer is thinking about implementing the current method, the other is thinking strategically about the whole system
  - Pairs are put together dynamically
  - http://www.cs.utah.edu/~lwilliam/Papers/ieeeSoftware.PDF
- Collective code ownership
  - Any programmer that sees an opportunity to add value to any portion of the code is required to do so at any time
- Continuous integration
  - Use of version and configuration management (e.g., CVS)
  - All changes are integrated into the code-base at least daily
  - The tests have to run 100% before and after the integration

XP Practices (IV)

- 40-h week
  - Programmers go home on time
  - Overtime is a symptom of a serious problem
  - No errors by tired developers; better motivated developers
- On-site customer
  - Development team has continuous access to a real life customer/user
- Coding standards
  - Everyone codes to the same standards
  - Ideally, you should not be able to tell by looking at it who has written a specific piece of code

XP Advantages

- Integrated, simple concept
- Low management overhead (no complicated procedures to follow, no documentation to maintain, direct communication, pair programming)
- Continuous risk management (early feedback from the customer)
- Continuous effort estimation
- Emphasis on testing; tests help in evolution and maintenance

XP Disadvantages

- Appropriate for small teams (up to 10 developers) only (does not scale)
- Large development groups may require more structures and documents
- If maintainers are not the people that developed the code, good documentation is necessary
- Generic design may be necessary to enable expected future development

Scrum

- "Scrum is not an acronym. It's an event in the game of rugby where like-minded people get together and politely discuss ownership of a ball."
- Rugby metaphor introduced by Takeuchi and Nonaka in their 1986 paper The New New Product Development Game that reported on innovative processes being used by big companies for new product development (e.g., cars, cameras, copiers.)
- Video: Ken Schwaber, Google Tech Talks
  - http://video.google.ca/videoplay?docid=-7230144396191025011&q=agile+development
  - Intro (2m00s to 9m27s).
  - Scrum works for any team (14m20s to 15m54s).
  - Game companies (33m00s to 35m13s)

©Todd Veldhuizen 2008

Scrum Process

Software Development Scrum

- A collection of practices, reacting against high-formality processes
- Key ideas:
  - Timeboxes with deliverables
  - Small, self-managed, cross-functional teams of 3-9 people
  - Daily meetings
  - ?

©Todd Veldhuizen 2008

Scrum meetings

- Daily, less than 15 minutes
- Each team member asked
  - What have you done since last meeting?
  - What has impeded your work?
  - What will you do next?
- Try to resolve impediments quickly, or schedule smaller meetings immediately following

©Todd Veldhuizen 2008

Scrum Sprint

- A short development interval, e.g. 3 weeks
- Well-defined goal: what’s going to be implemented, what’s going to be demoed at the end of the spring
- Keep track of *Backlog*
  - Prioritized list of work to be done (features, stories, requirements)
  - Rough estimate of ideal number of days required to implement each item
  - Before each sprint, a planning session is held to decide what items from the backlog will be addressed by the sprint

©Todd Veldhuizen 2008

End of Sprint

- End of sprint:
  - –Demo
  - –Collect feedback
  - –Discuss performance, process
  - –Plan next sprint

©Todd Veldhuizen 2008

Further Reading

- RUP
  - Philippe Kruchten. The Rational Unified Process: An Introduction (3rd Edition). Addison-Wesley, 2003
  - Craig Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, Prentice-Hall, 2002 (2nd edition)
  - Kendall Scott. The Unified Process Explained. Addison Wesley, 2001

Further Reading

- Agile development
  - – Kent Beck, Extreme Programming: Explained, Addison-Wesley, 1999
  - – R. Jeffries, C. Hendrikson, A. Anderson, Extreme Programming Installed, Addison-Wesley, 2001
    - http://member.netease.com/~wooce/tip/se/
  - – Ken Schwaber and Mike Beedle. Agile Software Development with Scrum. Prentice Hall, 2001
  - – Alistair Cockburn, Agile Software Development, Addison-Wesley 2002
  - – Pekka Abrahamsson, Juhani Warsta, Mikko T. Siponen, and Jussi Ronkainen. New directions on agile methods: a comparative analysis. In ICSE ’03: Proceedings of the 25th International Conference on Software Engineering, pages 244–254, Washington, DC, USA, 2003. IEEE Computer Society

Further Reading

- Online resources
  - www.extremeprogramming.org
  - http://www.xp123.com/
  - http://www.xprogramming.com
  - http://groups.yahoo.com/group/extremeprogramming
  - http://c2.com